**Taller Resuelto: Patrón Repository con .NET 8 Web API**

**Introducción**

El patrón Repository permite desacoplar la lógica de negocio del acceso a datos. En este taller, el estudiante implementará una API REST con .NET 8, aplicando buenas prácticas de arquitectura por capas y utilizando Entity Framework Core, AutoMapper y Swagger.

**Objetivos**

**Objetivo general**

Implementar una API Web con arquitectura por capas utilizando el patrón Repository con .NET 8, siguiendo principios SOLID y buenas prácticas.

**Objetivos específicos**

* Comprender la estructura de una solución con patrón Repository.
* Aplicar el uso de AutoMapper para mapear entidades con DTOs.
* Realizar pruebas con Swagger.
* Fortalecer el manejo de Entity Framework Core y la inyección de dependencias.

**Paso 1: Crear el proyecto**

1. Abre Visual Studio 2022.
2. Crea un nuevo proyecto tipo **ASP.NET Core Web API**.
3. Nombre: Repository.
4. Framework: **.NET 8.0**.
5. Asegúrate de que Swagger esté habilitado.
6. Da clic en “Crear”.

**Paso 2: Instalar paquetes NuGet**

Desde el **Administrador de paquetes NuGet**, instala:

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

AutoMapper

AutoMapper.Extensions.Microsoft.DependencyInjection

Swashbuckle.AspNetCore

**Paso 3: Crear estructura de carpetas**

Repository/

│

├── Controllers/

├── Data/

├── DTOs/

├── Models/

├── Profiles/

├── Repositories/

│ └── Implements/

├── Services/

│ └── Implements/

**Paso 4: Modelos (Entities)**

**Models/Student.cs**

namespace Repository.Models

{

public class Student

{

public int ID { get; set; }

public string LastName { get; set; } = string.Empty;

public string FirstMidName { get; set; } = string.Empty;

public DateTime EnrollmentDate { get; set; }

public ICollection<Enrollment>? Enrollments { get; set; }

}

}

**Models/Course.cs**

using System.Text.Json.Serialization;

namespace Repository.Models

{

public class Course

{

public int CourseID { get; set; }

public string Title { get; set; } = string.Empty;

public int Credits { get; set; }

[JsonIgnore]

public ICollection<Enrollment>? Enrollments { get; set; }

}

}

**Models/Enrollment.cs**

namespace Repository.Models

{

public enum Grade

{

A, B, C, D, E, F

}

public class Enrollment

{

public int EnrollmentID { get; set; }

public int CourseID { get; set; }

public int StudentID { get; set; }

public Grade Grade { get; set; }

public Course? Course { get; set; }

public Student? Student { get; set; }

}

}

**Paso 5: DbContext**

**Data/SchoolContext.cs**

using Microsoft.EntityFrameworkCore;

using Repository.Models;

namespace Repository.Data

{

public class SchoolContext : DbContext

{

public SchoolContext(DbContextOptions<SchoolContext> options) : base(options) { }

public DbSet<Student> students { get; set; }

public DbSet<Course> courses { get; set; }

public DbSet<Enrollment> enrollements { get; set; }

}

}

**Paso 6: DTOs**

**DTOs/StudentDTO.cs**

namespace Repository.DTOs

{

public class StudentDTO

{

public int ID { get; set; }

public string LastName { get; set; } = string.Empty;

public string FirstMidName { get; set; } = string.Empty;

public DateTime EnrollmentDate { get; set; }

}

}

**DTOs/CourseDTO.cs**

namespace Repository.DTOs

{

public class CourseDTO

{

public int CourseID { get; set; }

public string Title { get; set; } = string.Empty;

public int Credits { get; set; }

}

}

**Paso 7: AutoMapper**

**Profiles/AutoMapperProfile.cs**

using AutoMapper;

using Repository.Models;

using Repository.DTOs;

namespace Repository.Profiles

{

public class AutoMapperProfile : Profile

{

public AutoMapperProfile()

{

CreateMap<Student, StudentDTO>().ReverseMap();

CreateMap<Course, CourseDTO>().ReverseMap();

}

}

}

**Paso 8: Repositories**

**Repositories/IGenericRepository.cs**

namespace Repository.Repositories

{

public interface IGenericRepository<TEntity> where TEntity : class

{

Task<List<TEntity>> GetAll();

Task<TEntity> GetById(int id);

Task<TEntity> Insert(TEntity entity);

Task<TEntity> Update(TEntity entity);

Task DeleteById(int id);

}

}

**Repositories/IStudentRepository.cs**

using Repository.Models;

namespace Repository.Repositories

{

public interface IStudentRepository : IGenericRepository<Student>

{

}

}

**Repositories/ICourseRepository.cs**

using Repository.Models;

namespace Repository.Repositories

{

public interface ICourseRepository : IGenericRepository<Course>

{

}

}

**Paso 9: Repositories Implements**

**Repositories/Implements/GenericRepopsitory.cs**

using Microsoft.EntityFrameworkCore;

using Repository.Data;

namespace Repository.Repositories.Implements

{

public class GenericRepopsitory<TEntity> : IGenericRepository<TEntity> where TEntity : class

{

private readonly SchoolContext \_context;

public GenericRepopsitory(SchoolContext context)

{

\_context = context;

}

public async Task<List<TEntity>> GetAll() =>

await \_context.Set<TEntity>().ToListAsync();

public async Task<TEntity> GetById(int id)

{

var entity = await \_context.Set<TEntity>().FindAsync(id);

if (entity == null) throw new KeyNotFoundException();

return entity;

}

public async Task<TEntity> Insert(TEntity entity)

{

await \_context.Set<TEntity>().AddAsync(entity);

await \_context.SaveChangesAsync();

return entity;

}

public async Task<TEntity> Update(TEntity entity)

{

\_context.Set<TEntity>().Update(entity);

await \_context.SaveChangesAsync();

return entity;

}

public async Task DeleteById(int id)

{

var entity = await GetById(id);

\_context.Set<TEntity>().Remove(entity);

await \_context.SaveChangesAsync();

}

}

}

**StudentRepository.cs y CourseRepository.cs**:

// StudentRepository.cs

using Repository.Data;

using Repository.Models;

namespace Repository.Repositories.Implements

{

public class StudentRepository : GenericRepopsitory<Student>, IStudentRepository

{

public StudentRepository(SchoolContext context) : base(context) { }

}

}

// CourseRepository.cs

using Repository.Data;

using Repository.Models;

namespace Repository.Repositories.Implements

{

public class CourseRepository : GenericRepopsitory<Course>, ICourseRepository

{

public CourseRepository(SchoolContext context) : base(context) { }

}

}

**Paso 10: Services**

**Services/IGenericService.cs**

namespace Repository.Services

{

public interface IGenericService<TEntity> where TEntity : class

{

Task<List<TEntity>> GetAll();

Task<TEntity> GetById(int id);

Task<TEntity> Insert(TEntity entity);

Task<TEntity> Update(TEntity entity);

Task DeleteById(int id);

}

}

**IStudentService.cs / ICourseService.cs**

using Repository.Models;

namespace Repository.Services

{

public interface IStudentService : IGenericService<Student> { }

public interface ICourseService : IGenericService<Course> { }

}

**GenericService.cs**

using Repository.Repositories;

namespace Repository.Services.Implements

{

public class GenericService<TEntity> : IGenericService<TEntity> where TEntity : class

{

private readonly IGenericRepository<TEntity> \_repository;

public GenericService(IGenericRepository<TEntity> repository)

{

\_repository = repository;

}

public Task<List<TEntity>> GetAll() => \_repository.GetAll();

public Task<TEntity> GetById(int id) => \_repository.GetById(id);

public Task<TEntity> Insert(TEntity entity) => \_repository.Insert(entity);

public Task<TEntity> Update(TEntity entity) => \_repository.Update(entity);

public Task DeleteById(int id) => \_repository.DeleteById(id);

}

}

**StudentService.cs / CourseService.cs**

using Repository.Models;

using Repository.Repositories;

namespace Repository.Services.Implements

{

public class StudentService : GenericService<Student>, IStudentService

{

public StudentService(IStudentRepository repo) : base(repo) { }

}

public class CourseService : GenericService<Course>, ICourseService

{

public CourseService(ICourseRepository repo) : base(repo) { }

}

}

**Paso 11: Controllers**

**Controllers/StudentController.cs** y **CoursesController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Repository.Models;

using Repository.Services;

namespace Repository.Controllers

{

[Route("api/[controller]")] // Define la ruta base para este controlador, por ejemplo: /api/Course

[ApiController]

public class CoursesController : ControllerBase

{

private readonly ICourseService courseService;

public CoursesController(ICourseService courseService)

{

this.courseService = courseService;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Course>>> GetAll()

{

return await courseService.GetAll();

}

[HttpGet("{id}")]

public async Task<ActionResult<Course>> GetById(int id)

{

var course = await courseService.GetById(id);

if (course == null)

{

return NotFound($"Curso Id {id} no se encuentra");

}

return course;

}

// PUT: api/Course/5

// Para actualizar, el ID en la ruta (URL) debe coincidir con el CourseID del objeto en el cuerpo de la solicitud.

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, Course course)

{

if (id != course.CourseID)

{

return BadRequest("El ID de la URL no coincide con el ID del curso proporcionado.");

}

try

{

await courseService.Update(course);

}

catch (DbUpdateConcurrencyException)

{

if (!await CourseExists(id))

{

return NotFound($"Curso con ID {id} no encontrado.");

}

else

{

throw; // Otra excepción de concurrencia

}

}

return NoContent(); // Indica que la operación fue exitosa pero no hay contenido que devolver

}

[HttpPost]

public async Task<ActionResult<Course>> Insert(Course course)

{

await courseService.Insert(course);

return CreatedAtAction(nameof(GetById), new { id = course.CourseID }, course);

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteById(int id)

{

await courseService.DeleteById(id);

return NoContent();

}

private async Task<bool> CourseExists(int id)

{

var course = await courseService.GetById(id);

return course != null;

}

}

}

using Microsoft.AspNetCore.Mvc;

using AutoMapper;

using Repository.DTOs;

using Repository.Models;

using Repository.Services;

namespace Repository.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class StudentController : ControllerBase

{

private readonly IStudentService \_studentService;

private readonly IMapper \_mapper;

public StudentController(IStudentService studentService, IMapper mapper)

{

\_studentService = studentService;

\_mapper = mapper;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<StudentDTO>>> GetAll()

{

var students = await \_studentService.GetAll();

var studentDTOs = \_mapper.Map<IEnumerable<StudentDTO>>(students);

return Ok(studentDTOs);

}

[HttpGet("{id}")]

public async Task<ActionResult<StudentDTO>> GetById(int id)

{

var student = await \_studentService.GetById(id);

if (student == null) return NotFound();

var studentDTO = \_mapper.Map<StudentDTO>(student);

return Ok(studentDTO);

}

[HttpPost]

public async Task<ActionResult<StudentDTO>> Create(StudentDTO studentDTO)

{

var student = \_mapper.Map<Student>(studentDTO);

var newStudent = await \_studentService.Insert(student);

return Ok(\_mapper.Map<StudentDTO>(newStudent));

}

[HttpPut("{id}")]

public async Task<ActionResult<StudentDTO>> Update(int id, StudentDTO studentDTO)

{

if (id != studentDTO.ID) return BadRequest();

var student = \_mapper.Map<Student>(studentDTO);

var updatedStudent = await \_studentService.Update(student);

return Ok(\_mapper.Map<StudentDTO>(updatedStudent));

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

await \_studentService.DeleteById(id);

return NoContent();

}

}

}

**Paso 12: Pruebas con Swagger**

Ejecuta el proyecto (F5 o Ctrl + F5). Swagger abrirá en el navegador y podrás probar los endpoints para:

* Crear estudiantes/cursos.
* Consultarlos.
* Actualizarlos.
* Eliminarlos.

**Conclusión**

Este taller mostró cómo aplicar el patrón Repository en una arquitectura por capas, reforzando buenas prácticas como la inyección de dependencias, el uso de AutoMapper, y la separación de responsabilidades. Es una excelente base para proyectos más complejos y reales.